The Next 10^4 UppSAT Approximations

Aleksandar Zeljić[1], Peter Backeman[1],
Christoph M. Wintersteiger[2], and Philipp Rümmer[1]

[1] Uppsala University, Uppsala, Sweden
name.surname@it.uu.se
[2] Microsoft Research, Cambridge, UK
cwinter@microsoft.com

Abstract

Reasoning about complex SMT theories is still quite challenging, for instance theories of bit-vectors, floating-point arithmetic, and strings. Approximations offer a means of mapping a complex theory into a simpler one, and attempting to reconstruct models or proofs in the original theory afterwards. UppSAT is an approximating abstract SMT-solver [1], based on the systematic approximation refinement framework [9]. The framework can be instantiated using an approximation and a back-end SMT solver. Implemented in Scala, UppSAT is designed with easy and flexible specification of approximations in mind. We discuss the structure of approximations in UppSAT and the components needed for their specification. Because approximation components can be defined relatively independently, they can be flexibly combined to obtain many different flavours of approximation. In this extended abstract we discuss what kinds of approximations can be expressed in UppSAT, along with design choices that enable the modular *mix-and-match* specification of approximations. Finally, we also outline ideas for several new approximations and strategies which we are currently working.

1 Introduction

The construction of satisfying assignments (i.e., models) of a formula, or showing that no such assignments exist, is at the heart of Satisfiability Modulo Theories (SMT). It is a field with extensive research, but there are still theories for which effective model construction is challenging. Such theories are, in particular, non-linear arithmetic domains such as bit-vectors, non-linear real arithmetic (or real-closed fields), and floating-point arithmetic; even when decidable, the high computational complexity of such problems turns model construction into a bottleneck in applications such as model checking, test-case generation, or hybrid systems analysis.

In several recent papers [1, 9], the notion of *approximation* has been proposed as a means to speed up the construction of (precise) satisfying assignments. Generally speaking, approximation-based solvers follow an iterative strategy to find a satisfying assignment of a formula ϕ: First, a simplified or *approximated* version $\hat{\phi}$ of ϕ is solved, resulting in an approximate solution $\hat{m}$ that (hopefully) lies close to a precise solution (for some measure of closeness). Second, a *reconstruction* procedure is applied to check whether $\hat{m}$ can be turned into a precise solution m of the original formula ϕ. If no precise solution m close to $\hat{m}$ can be found, *refinement* is used to successively obtain better, more precise, approximations.

This high-level approach opens up a large number of design choices, some of which have been discussed in the literature. The approximations considered have different properties; for instance, they might be over- or under-approximations (in which case they are commonly called *abstractions*), or be non-conservative and exhibit neither of those properties. The approximated formula $\hat{\phi}$ can be formulated in the same logic as ϕ, or in some *proxy* theory that enables more efficient reasoning. The reconstruction of m from $\hat{m}$ can follow various strategies, including

simple re-evaluation, precise constraint solving on partially evaluated formulas, or randomised optimisation. Refinement can be performed with the help of approximate assignments $\hat{m}$, using proofs or unsatisfiable cores, or be completely independent of the reason of the failure.

In this extended abstract, we present a brief overview of UppSAT, an approximating abstract SMT-solver. We focus on practical aspects and present current research directions which are particularly relevant for the SMT community. We refer the reader for a more comprehensive description of the abstract framework [9] and UppSAT [1].

2 UppSAT

Approximation anatomy. At the heart of the approximation scheme outlined in Figure 1 is the translation of a formula ϕ_{in} of input theory T_{in} to a formula ϕ_{out} of theory T_{out}. To control the granularity of the approximation, the translations are dependent on some *precision*—a partially ordered domain which satisfy the ascending chain condition (every ascending chain is finite) [9]. We refer to the pair of theories T_{in} and T_{out} together with the precision as the approximation *context*. Fixing the context only frames the approximation but does not define it. This is done by the *codec*, which consists of two parts: a precision-dependent translation (the *encoding* of constraints of the input theory into constraints of the output theory), and a translation of models of the output theory to models of the input theory (the *decoding*). The goal of the encoding is to translate into a set of constraints that are simpler to solve by some measure. One should note that the encoding can be a translation from a theory unto itself, using a simplifying translation.

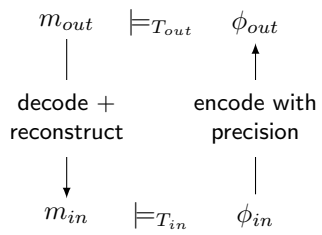

Figure 1: Commutativity graph showing how to obtain the model m_{in} via an output theory T_{out}.

Once an approximated formula ϕ_{out} is obtained, it is sent to a suitable back-end solver. This solver returns attempts to solve ϕ_{out}, and it produces either SAT or UNSAT as its result. If the result is UNSAT, a proof of unsatisfiability or an unsat core can usually be extracted and we may attempt to generalize and rewrite them such that they apply to the input formula ϕ_{in}. The current version of UppSAT, however, does not attempt this and instead naïvely increases the precision, to create a more refined approximation. (Conveniently, this is also a viable way to handle undecidable theories and incomplete back-end solvers that return UNKNOWN for some queries.)

If the result is SAT, an approximate model m_{out} is obtained. This model is decoded to a candidate model m_{in}, which may or may not be an actual model of the input formula ϕ_{in}. If it is, then m_{in} is returned as a solution and the procedure terminates. If it is not, model *reconstruction* is applied – a procedure which attempts to slightly modify m_{in} into a new candidate model $\hat{m}_{in}$ which is once again checked against the input formula ϕ_{in}. If it is still not satisfied, *model-based refinement* is activated, which increases precision based on the approximate model m_{out} and candidate model m_{in}.

Precision refinement, both in the UNSAT and the SAT case ranges from a naïve, uniform increase of precision of each symbol of the formula, to a more fine-grained refinement where only certain variables and operators have their precision increased (non-uniformly). The whole process is illustrated in Figure 2.

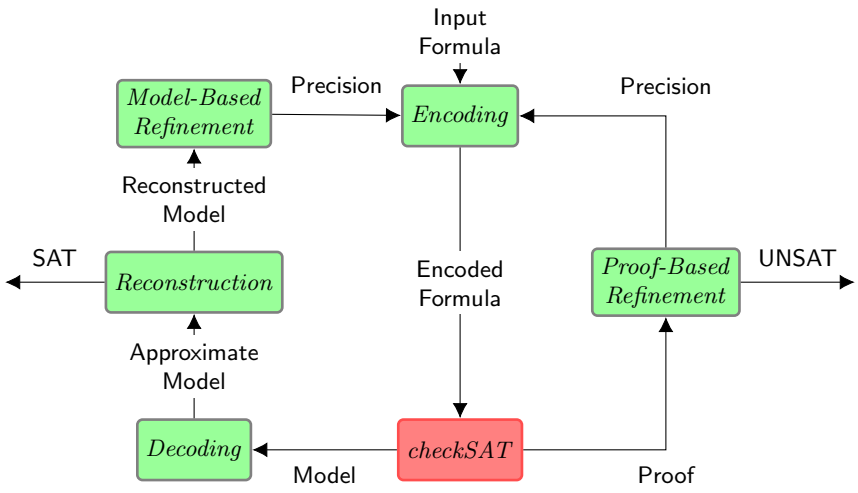

Figure 2: The main components needed to implement approximations in UppSAT, and the flow of data in between them.

Goals of Approximations. The core idea of using approximations is to abstract away complexity while retaining the essence of the constraints, e.g., by reducing the search space or simplifying semantics. There are a number of ways this can be achieved:

- Reducing the search space by adding constraints to the formula, e.g., limiting the length of a string or an array, or imposing interval constraints on numeric data types.
- Reducing the search space by going to smaller data type, e.g., in case of parametric data types this is achieved by re-typing the variables and constraints.
- Removing challenging semantics such as non-linear constraints or rounding operations, by moving to theories that are simpler but retain the relevant aspects, e.g., moving from floating-point arithmetic (FPA) to bit-vector arithmetic (BVA).

Power of UppSAT In previous work the approximation framework and the approximation were implemented within a SMT-solver [9]. Implementation of a new approximation required a lot of code duplication and restricted the choice of decision procedures. In UppSAT, the approximation framework is implemented as an abstract SMT solver, which is instantiated using an approximation and an SMT-solver as a back-end. By using UppSAT, there are several advantages:

- It is easy to support new SMT-solvers as back-ends, enabling the comparison of an approximation against different solving algorithms and their implementations.
- UppSAT comes with pre-defined components and templates, allowing for easy and compact specification of new approximations. Of course, new templates and components can also be added.

- UppSAT enables new non-approximating SMT solvers, by simply implementing a precise translation between the theories. For example, using a bit-precise translation of FPA into BVA and a bit-vector back-end, yields an SMT solver for FPA. This is also a simple way to resolve combination of FPA and BVA, by translating it into BVA upfront.

In Table 1 (taken from [1]) a comparison of back-ends and instantiations of the UppSAT framework are presented on non-trivial satisfiable floating-point formulas of the SMT-LIB (QF_FP). It shows that there is significant performance improvement to be had using the approximation approach. The SMT solvers MathSAT [4], Z3 [5], and Sonolar [7] feature bit-precise conversions from FPA to bit-vector constraints, known as bit-blasting, and represent the currently most commonly used solvers in program verification. An alternative, constraint programming-based approach to solve FPA constraints is implemented in COLIBRI [2]. Currently, only Z3 and MathSAT are supported as back-ends by UppSAT.

	ACDCL	MathSAT	Z3	FPA ↔ FXA (Z3)	SmallFloat (ACDCL)	SmallFloat (MathSAT)	SmallFloat (Z3)
Solved	86	99	97	91	78	**101**	**101**
Timeouts	44	31	33	39	52	29	29
Best	65	4	6	9	3	9	9
Iterations	-	-	-	2.69	3.59	3.16	3.02
Only solver	1	0	2	0	0	1	0

Table 1: Back-ends and instantiations of UppSAT, showing # of benchmarks solved within 1 hour, # of timeouts, # of instances for which the solver was fastest, average # of refinement iterations on solved problems, and # of instances only solved by the respective solver.

10^4 Approximations. When defining an approximation in UppSAT, there are several key choices to be made. These are illustrated in Table 2. First of all, an input and and a output theory are chosen, defining the **context**. Secondly, a **codec** is picked, defining the semantics of precision, encoding and decoding. This can be, for example, a monolithic encoding where the precision regulates the encoding of the entire formula, or a composite one where precision is assigned to each variable and operation in the formula, encoding them independently of each other (modulo well-sortedness). Other examples include reducing the size of floating-point types, described as SmallFloat in [9], or translating from floating-point arithmetic to fixed-point arithmetic (FXA) as described in Section 3. Thirdly, a **reconstruction** strategy must be chosen, for instance, this can be equality-as-assignment, a heuristic using equalities to propagate values (see [1]), or evaluation-based using a back-end to patch failing constraints such as numeric model lifting [8]. Finally, a **refinement** strategy should be decided, which defines how precision should be increased when reconstruction fails, this can be a simple uniform increase or using the discrepancy between candidate model and the reconstructed model to decide which precision to increase. Furthermore, a notion of error can be used to decide by how much to increase the precision.

Many of these choices are orthogonal, which enables a multitude of approximations to be composed from a few distinct components. Together with the rapid prototyping of UppSAT, it allows after the introduction of a new component to quickly try many new approximations with relatively little coding effort. For example, re-implementation of the SmallFloat approximation takes fewer than 300 lines of commented Scala code.

Context	Codec	Reconstruction	Refinement
$(T_{in} \times T_{out}, \mathcal{P})$	Monolithic Encoding SmallFloat FPA $\leftrightarrow$ FXA $\vdots$	Evaluation-Based Equality-as-Assignment Numerical Model Lifting $\vdots$	Uniform Error-Based $\vdots$

Table 2: Choices when defining approximations

3 Work In Progress

We are currently working on a number of different directions, where the goal is to extend and test the capabilities of UppSAT.

Local Search Reconstruction. Local search strategies are often employed in optimisation and there are adaptations of such for the SAT problem (see e.g., [6]). Currently, UppSAT model reconstruction is done using a linear pass (with equality-as-assignment), where a failed model is passed through once and heuristics are applied to it in an attempt to turn it into a model. However, this often fails and after refinement another iteration is performed. Since UppSAT spends most of its runtime finding approximate models, there is plenty of room for effective use of resources on a more extensive portfolio of model reconstruction strategies.

The assumption is that a candidate model lies close (if not equal) to a model of the input formula. Therefore, using local search to probe 'similar' models, or models 'close' to the approximate one (by some measure), intuitively makes sense. The challenge lies in defining a good measure of fitness, such as to steer the search toward finding better and better models, for some notion of 'better'. For example, in numerical domains, a value that depends on how large the violation is, e.g., the absolute difference between two numbers which should be equal, could be used. By minimizing the sum of these violations, a search direction is defined such that when fitness is zero an actual model is found.

Improved Fixed-Point Approximation. In a previous paper [1], a fixed-point approximation for floating-point numbers was presented as a proof of concept. We are currently extending this work, primarily by introducing a more sophisticated refinement method. In the naïve approximation, floating-point number were translated into fixed-point numbers based on a precision p, such that the encoded numbers had p integral bits and p fractional bits. A uniform refinement strategy was used, i.e., whenever model reconstruction failed, the precision for the whole formula was uniformly increased. Now we propose to improve this ordering by:

- Use a composite precision for encoding, enabling a fine-grained encoding and reconstruction.
- Use a vector precision domain (i, f) making the number of integral bits i and fractional bits f independent from each other.
- Refine precision based on where and how the reconstruction failed, e.g., by studying how many bits were lost in encoding.

Experiments using a naïve implementation thereof show the potential of these changes having a significant impact on performance and that they may well result in a very competitive decision procedure for floating-point arithmetic.

Linear Approximation of Non-linear Constraints. There exist many efficient (decision) procedures for solving linear real constraints. However, when dealing with non-linear formulas, the problem gets much harder. We propose an approximation of FPA using linear constraints over reals to employ existing, efficient decision procedures.

A straightforward, but naïve, method is to simply guess the values of some of the involved variables, thus cutting out constraints over non-linear operators (akin to theory decisions in SMT context). If the approximation is unsatisfiable, chosen approximate values will be adjusted. If an approximate model is found, then reconstruction is attempted which would yield better values for the next iteration if the reconstruction fails. More interestingly, it might be possible to use approximate models and the Taylor-expansion around the approximate values to iteratively obtain better estimates of higher-order factors. The inspiration comes from work on incremental linearization of transcedental functions by Cimatti et al. [3].

4 Conclusion

We present the UppSAT framework and describe how it can be used for effective construction of approximating decision procedures. We show the anatomy of UppSAT approximations and observe that there is a large space of approximations available to be explored. Furthermore, we propose that UppSAT is a great test-bed environment to explore these approximations with little implementation overhead. We also introduce three currently ongoing research projects, the first extending UppSAT library of components with a local search algorithm for model reconstruction, and two projects focused on using UppSAT to solve problems within theories which current solvers are struggling to deal with efficiently.

References

[1] Peter Backeman, Aleksandar Zeljić, Christoph M. Wintersteiger, and Philipp Rümmer. Exploring approximations for floating-point arithmetic using UppSAT. In *Automated Reasoning, 9th International Joint Conference (IJCAR)*, LNCS. Springer, 2018. To appear.

[2] Franois Bobot, Zakaria Chihani, and Bruno Marre. Real behavior of floating point. In *15th International Workshop on Satisfiability Modulo Theories (SMT 2017)*, 2017.

[3] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Satisfiability modulo transcendental functions via incremental linearization. *CoRR*, abs/1801.08723, 2018.

[4] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT solver. In *TACAS*, volume 7795 of *LNCS*, 2013.

[5] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*. Springer, 2008.

[6] Holger H. Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *J. Autom. Reason.*, 24(4):421–481, May 2000.

[7] F. Lapschies, J. Peleska, E. Gorbachuk, and T. Mangels. SONOLAR SMT-solver. In *SMT-COMP system description*, 2012.

[8] Jaideep Ramachandran and Thomas Wahl. Integrating proxy theories and numeric model lifting for floating-point arithmetic. In *FMCAD*. IEEE, 2016.

[9] Aleksandar Zeljic, Christoph M. Wintersteiger, and Philipp Rümmer. An approximation framework for solvers and decision procedures. *JAR*, 58(1):127–147, 2017.